Evaluating a DVS Scheme for Real-Time Embedded Systems *

Ruibin Xu, Daniel Mossé, Rami Melhem
Computer Science Department, University of Pittsburgh
{*xruibin,mosse,melhem*}*@cs.pitt.edu*

Abstract

Dynamic voltage scaling (DVS) has become a well-known and effective technique to exploit energy-performance trade-off in real-time embedded systems where energy imposes a major constraint. We focus on frame-based real-time systems that execute variable workloads with the goal of minimizing expected energy consumption in the system while still meeting the deadlines. In our separate publication, we proposed a new DVS scheme that incorporates the dynamic behavior of the tasks into the speed schedule and aims to minimize the expected energy consumption in the system. The new DVS scheme was derived based on the assumption of unrestricted continuous frequency. However, it remains unknown how the new DVS scheme performs in practical situations. In this paper, we first give a simple example through which we demonstrate the new DVS scheme and compare it with the existing DVS schemes. Then we present evaluation results to show that the new DVS scheme achieves significant energy savings over the existing schemes.

1 Introduction

Energy conservation is critically important for many real-time systems such as battery-operated embedded systems which have a restricted energy budget . Dynamic voltage scaling (DVS), which involves dynamically adjusting the voltage and frequency of the CPU, has become a well-known technique in power management for real-time embedded systems. Through DVS, quadratic energy savings can be achieved at the expense of just linear performance loss [14, 4]. Thus, the execution of tasks can be slowed down in order to save energy, as long as the deadline constraints are not violated. A natural problem that rises from this context is how to minimize the energy consumption in the system while still meeting the deadlines. The problem is often reduced to determining a task's speed (or equivalently, determining the amount of time allotted to a task) before it is scheduled to execute in the system.

The systems under our consideration are frame-based hard real-time embedded systems that execute variable workloads. The tasks in these systems exhibits dynamic behavior in the sense that they usually run for less than their worst-case execution times (WCET) and the execution time of the tasks is unpredictable before their execution. Therefore, the design goal of DVS schemes becomes *minimizing the expected (total) energy consumption* in the system.

In [12], we proposed a new DVS scheme that incorporates the dynamic behavior of the tasks into the speed schedule and aims to minimize the expected energy consumption in the system. To our knowledge, this is the first DVS scheme that is designed explicitly to Minimize the Expected Energy Consumption for frame-based real-time systems. Therefore, we call the new DVS scheme MEEC throughout this paper. The MEEC scheme was derived based on the assumption of unrestricted continuous frequency. We also extended it to take into consideration the practical issues, such as minimum and maximum frequency restriction, and provided solutions to the problems. However, it remains unknown how the MEEC scheme performs under all the practical considerations. In this paper, we first give a simple example through which we demonstrate the MEEC scheme and compare it with the existing DVS schemes. We show that failure to capture the dynamic behavior of the tasks by the existing DVS schemes and naive use of dynamic behavior information will lead to suboptimal power management. Then we present extensive evaluation results, both on synthetic and real-life workloads, to show that the MEEC scheme can achieve significant energy savings over the existing schemes.

This paper is organized in the following way. We first present the related work in Section 2. The system and task model are described in Section 3. We demonstrate the MEEC scheme and compare it with the existing schemes in Section 4. Evaluation results are presented in Section 5. We end the paper in Section 6 with concluding remarks.

*This work has been supported by NSF grant ANI-0125704 and ANI-0325353.

2 Related Work

Although much work has been done on exploring DVS in real-time environments, we will focus on the related work that takes into consideration actual (not worst-case) execution time of tasks. This is because real-time applications usually exhibit a large variation in actual execution times (e.g., [3] reported that the ratio of the worst-case execution time to the best-case execution time can be as high as 10 in typical applications; our measurements in [10] show that this ratio can be as high as 100), and thus the DVS schemes that use exclusively worst-case execution time lack the advantage of unused computation time. Besides frame-based real-time systems, we will also focus on the related work that applies to periodic real-time systems because frame-based real-time system is a special case of periodic real-time system.

DVS in real-time applications is categorized as *inter-task* or *intra-task* voltage scaling [5]. Inter-task schedules speed changes at each task boundary, while intra-task schedules speed changes within a single task. For inter-task voltage scaling, Mossé et al. [8] introduced the concept of *speculative speed reduction* and proposed three DVS schemes with different speed reduction aggressiveness for frame-based real-time systems. Aydin et al. [2] and Pillai et al. [9] independently proposed DVS schemes for achieving high energy savings for periodic real-time systems. They both precompute a static optimal schedule assuming that each task runs for WCEC and when a task runs for less than its WCEC, the scheduler uses the slack to create a new schedule for the remaining tasks. However, the exclusive use of static information in computing speed schedules by [8, 2, 9] leads to suboptimal power management for the system. The MEEC scheme makes use of both static and dynamic information to design the speed schedule. To be able to navigate the full spectrum of speculative speed reduction, in [2] system designers can set a parameter to control the degree of speed reduction aggressiveness. The MEEC scheme chooses the degree of speed reduction aggressiveness automatically, based the probability distribution of the workload of the tasks, to minimize the expected energy consumption.

For intra-task voltage scaling, Lorch et al. [6] have shown that if a task's computational requirement is only known probabilistically, there is no constant optimal speed for the task and the expected energy consumption is minimized by gradually increasing speed as the task progresses, which is an approach named as *Processor Acceleration to Conserve Energy* (PACE). Practical PACE (PPACE) [13] takes into consideration a number of practical issues and improves the performance of PACE. However, PACE and PPACE have only been studied for single task when considering hard real-time guarantee. In [7], PACE is used for soft real-time systems when the system has only one task but the maximum speed is used when the system has multiple tasks. In [12], we presented the theoretical results of using PACE for multiple tasks with a single hard deadline (frame length). We also show that a naive extension of PACE for multiple tasks is not recommended in Section 4.

AbouGhazaleh et al. [1] proposed a hybrid compiler-operating system intra-task DVS scheme for energy consumption of time-sensitive embedded applications. The MEEC scheme is implemented at the operating system level and assumes no access to application source codes.

3 Task and System Model

We consider a frame-based task model with N periodic tasks in the system, all ready at time zero. The task set is denoted by $T = \{T_1, T_2, \ldots, T_N\}$. Each task T_i ($1 \leq i \leq N$) is characterized by its worst-case execution cycles (WCEC) W_i and the probability density function of its execution cycles $P_i(x)$, which denotes the probability that task T_i executes for x ($1 \leq x \leq W_i$) cycles. Obviously, we have $\sum_{x=1}^{W_i} P_i(x) = 1$ and $P_i(W_i) \neq 0$. The average-case execution cycles (ACEC) of T_i can be computed as $\sum_{x=1}^{W_i} P_i(x)x$. All task periods are identical and all task deadlines are equal to their period. The common deadline/period (also known as frame length) is denoted by D. The execution of the frame is to be repeated and all tasks must be executed during each frame in the order of $T_1, T_2, \ldots, T_N$. Thus, the tasks can be treated as sequential sections of a single application. If the execution order of the tasks is flexible, the ordering strategies can be found in [12].

The tasks are to be executed on a variable voltage processor with the ability to dynamically adjust its frequency and voltage on application requests. Because processor is the major power consumer for many embedded systems, reducing processor energy consumption has a significant impact on the overall system energy consumption. In deriving the MEEC scheme [12], we assume that the processor frequency can be adjusted continuously from 0 to infinity. We also discuss the more realistic cases, such as the processor has minimum and maximum frequencies, in [12]. The processor power consumption when running at frequency f is $c_0 + c_1 f^\alpha$ (α is a constant that is at least 2) where c_0 and c_1 denote the power consumption of the processor when idle and the maximum dynamic power respectively. The dynamic power is determined by the processor operating frequency and the maximum dynamic power is the dynamic power consumed when the processor is operating at the maximum frequency.

4 The DVS Schemes

In this section, we give a simple example through which we demonstrate the MEEC scheme and compare it with the

Table 1. The parameters for the 3 tasks in the simple example

Task	W	P(x)	ACEC
T_1	2	$.9, .1$	1.1
T_2	4	$.9, 0, 0, .1$	1.3
T_3	2	$.5, .5$	1.5
$\hat{T}$	8	$0, 0, .405, .45, .045, .045, 0.05, .005$	3.9

existing schemes.

Example Suppose that there are 3 tasks in the frame-based real-time system with a frame length of 14 time units. The workload of a task in expressed in *super cycles*. A super cycle consists of a certain number of CPU cycles, which can be computed in order to keep the overhead of DVS low [1]. The tasks are required to be executed in the order of T_1, T_2, and T_3. The parameters for the 3 tasks are shown in Table 1. We also treat the three tasks as the three sequential sections of a single task $\hat{T}$ and its parameters are computed from those of the 3 tasks. $\hat{T}$ is used for the naive extension of PACE shown at the end of this section. For the processor, we suppose that $c_0 = 0$ and $c_1 = 1$. The maximum speed of the processor is 1 super cycle per time unit and the minimum speed of the processor is 0.

We start by reviewing the existing DVS schemes, which can be categorized into 3 schemes: proportional scheme, greedy scheme, and statistical scheme. The proportional scheme and greedy scheme only make use of WCEC and deadline information. The proportional scheme distributes the slack proportionally among all unexecuted tasks. Thus, in the example, the proportional scheme will start executing T_1 using speed $\frac{2+4+2}{14} = 0.5714$. The greedy scheme is more aggressive, because it gives all the slack to the next ready-to-run task. Therefore, the greedy scheme will start executing T_1 using speed $\frac{2}{14-(4+2)/1} = 0.25$. Note that the greedy scheme is using the lowest possible speed to execute the next task. The statistical scheme tries to take advantage of the average-case execution cycles (ACEC) of the tasks, to distribute the reclaimed slack, the natural slack, and the slack that would appear in the system if other tasks were to finish early. To guarantee that the deadline is not missed, the statistical scheme chooses the maximum of the speed obtained from the greedy scheme and the speed computed based the ACEC of the tasks. Thus, the statistical scheme will start executing T_1 using speed $max(0.25, \frac{1.1+1.3+1.5}{14}) = 0.2786$.

After a task finishes, the system reclaims the slack created by the task if it runs for less than its WCEC, and compute the speed of the next task recursively. This is also a common part of all dynamic-claiming DVS schemes, as follows. Let us see how they compute the speed for T_2 after T_1 finishes. Suppose that T_1 only runs for 1 super cycle. Then the time left for executing T_2 and T_3 in the proportional scheme is $14 - \frac{1}{0.5714} = 12.2499$, and speed $\frac{4+2}{12.2499} = 0.4898$ will be used to execute T_2. Similarly, the greedy scheme will use speed $\frac{4}{14-1/0.25-2/1} = 0.5$ to execute T_2, and the statistical scheme will use speed $max(\frac{4}{14-1/0.2786-2/1}, \frac{1.3+1.5}{14-1/0.2786}) = 0.4756$ to execute T_2.

Intuitively, when tasks tend to run close to their WCECs, the proportional scheme would perform well; when tasks tend to run much less than their WCECs, the greedy scheme would have good performance. The statistical scheme tries to strike a balance between proportional scheme and greedy scheme. However, none of them is optimal in terms of minimizing the expected energy consumption in the system.

The MEEC scheme incorporates the dynamic behavior of the tasks into the speed schedule. The dynamic behavior of the tasks is captured by the probability density function of the workload of the tasks, which is represented by histograms in practice. When using profiling to obtain WCEC and ACEC, the probability density function of the workload of the tasks can be also learned at the same time, only requiring certain amount of additional storage.

DVS Algorithm The MEEC scheme is divided into two phases: (a) the offline phase precomputes the speed schedule, which consists of the percentage factor β_i for each task T_i. The percentage factor β_i determines the speed to execute T_i: when T_i is ready to execute and the time left in the frame to execute $T_i, T_{i+1}, \ldots, T_N$ is d, then time $\beta_i d$ is allocated to execute T_i. The algorithm computes the percentage factors in the reverse order, that is, first compute β_N, then β_{N-1}, ..., and last β_1. The value of β_N is always 100% and the other percentage factors can be computed recursively and efficiently thanks to the nice property of energy function of the tasks. More details can be found in [12]; (b) the online phase is invoked before the execution of each task, obtaining the time left in the frame and computing the execution speed for the task: when starting executing task T_i and having time d left, set the speed to $\frac{W_i}{\beta_i d}$ (the actual speed value needs to be adjusted according to the available discrete speeds of the processor). Both phases are efficient: the offline phase runs in polynomial time and the online phase only takes constant time.

In the example, the percentage factors for T_1, T_2, T_3 can be computed to be equal to $39.38\%, 76.19\%, 100\%$, respectively. Thus, the MEEC scheme will use speed $\frac{2}{39.38\% \times 14} = 0.3628$ to execute T_1. If T_1 runs for 1 super cycle, then the time left for executing T_2 and T_3 is $14 - \frac{1}{0.3628} = 11.2737$. Then the MEEC scheme will use speed $\frac{4}{76.19\% \times 11.2737} = 0.4657$ to execute T_2. Table 3 shows the expected energy consumption per frame for all DVS schemes and the savings of the MEEC scheme over the other schemes. In [12], we prove that the MEEC scheme minimizes the expected energy consumption in the system under the assumption of

Table 2. The comparison of all DVS schemes for the simple example

Scheme	Expected energy consumption per frame	Saving
naive PACE	0.7953	23%
proportional	0.7733	21%
greedy	0.7388	17%
statistical	0.6771	10%
MEEC	0.6097	–

unrestricted continuous frequency.

Finally, we show through the simple example that a naive extension of PACE (or, naive PACE for short) cannot obtain energy savings over the DVS schemes that do not use intra-task voltage scaling. Since PACE has only been studied for a single task, the naive PACE treats all the tasks as a single super task and derives its parameters (WCEC and probability distribution of the workload) from those of the original tasks. For the example, the parameters for the super task $\hat{T}$ are shown in Table 1. For this super task, using PACE [6] will result in expected energy consumption per frame of 0.7953, which is the worst of all DVS schemes discussed so far. The reason why the naive PACE fails is that treating all tasks as a single super task results in loss of information (e.g., the naive PACE cannot determine when tasks terminate), losing the opportunity for dynamic slack reclamation. For instance, if task T_1 runs only for 1 super cycle, we can be sure that the rest of workload in the current frame is at most 6 super cycles. However, the naive PACE still assumes that the rest of workload is 7 super cycles in the worst case. In [12], we show that PACE must be used for executing individual tasks in order to obtain further energy savings over the DVS schemes that do not use intra-task voltage scaling.

5 Evaluation

The optimality of the MEEC scheme [12] only holds if we assume unrestricted continuous frequency which does not hold in practice. Therefore, we also discuss the issues that arise when our DVS scheme is used in practice and provide solutions to the problems in [12]. However, it remains unknown how the MEEC scheme performs under those practical considerations. To answer this question, we conducted extensive simulations for different power models and different workloads.

5.1 Power Models

We used two power models in our simulation. The first power model is a synthetic processor that strictly conforms to the $p(f) = f^3$ power-frequency relation and has 10 discrete frequencies ranging from 100MHz to 1000MHz with

Table 3. XScale speed settings and power consumptions

Speed (MHz)	150	400	600	800	1000
Voltage (V)	0.75	1.0	1.3	1.6	1.8
Power (mW)	80	170	400	900	1600

100MHz step; its idle power is zero. The second power model is the Intel XScale (Table 3) [11]. For the idle power of Intel XScale, we assume that the CPU operates at the lowest frequency (i.e., 150 MHz) when idle. This is equivalent to say that the idle power is 80 mW. The power function for XScale used in deriving β_i's is

$$p(f) = 80 + 1520(\frac{f}{1000})^3 \quad (1)$$

where f is the frequency. Figure 1 shows that Equation (1) is a good approximation of the actual power function of Intel XScale.

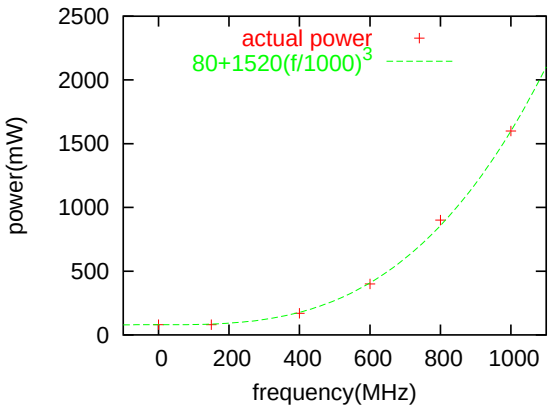
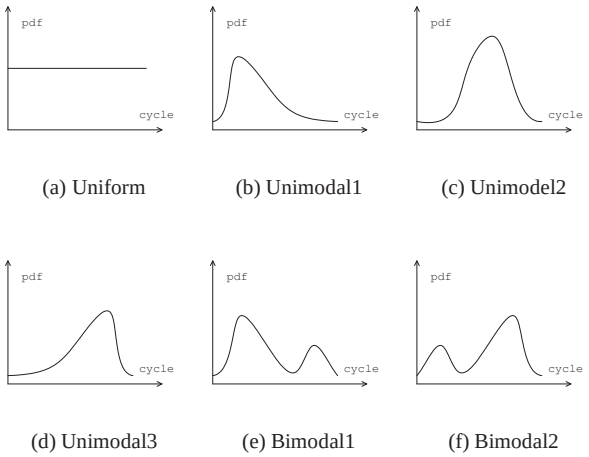


Figure 1. Approximate power function for Intel XScale

5.2 Synthetic Workloads

A frame-based real-time systems is characterized by the number of tasks, the WCEC of each task, the probability distribution of the workload of each task, the frame length. We simulated system that have 5, 10, 15, 20 tasks, respectively. We only show the results for the systems with 5 tasks because the results for systems with different number of tasks are similar. The WCEC of each task is randomly generated from 10,000,000 cycles to 1,000,000,000 cycles. The probability density function of each task's actual execution cycles is randomly chosen from 6 representative distributions shown in Figure 2. The bin width of the histograms denoting the probability density functions is 1,000,000 cycles. For each combination of the tasks, we computed the

worst-case finishing time (t) for a frame running at the highest speed. Then we varied the frame length from $1.2t$ to $4t$. For each simulated system (i.e., for each run with a set of tasks), we evaluated 8 DVS schemes: proportional without PACE (P), proportional with PACE [1] (PP), greedy without PACE (G), greedy with PACE (GP), statistical without PACE (S), statistical with PACE (SP), MEEC without PACE (M), MEEC with PACE (MP). For each experiment, we generated 100,000 frames and computed the average energy consumption per frame for each scheme. Under this experimental setup, we conducted over one million runs and averaged the results (which are shown here).



(a) Uniform (b) Unimodal1 (c) Unimodel2

(d) Unimodal3 (e) Bimodal1 (f) Bimodal2

Figure 2. Candidate probability density functions

For all the simulations using the synthetic CPU, the best scheme is always scheme M or MP, but scheme MP is only better than scheme M for 13.6% of the time with an average saving of 1.2% over scheme M. For all the simulations using XScale, the best scheme is always scheme M. Note that, in the simulations, we ignore the speed change overhead and online scheduling overhead, and thus we favor schemes using PACE. For the other schemes, scheme PP outperforms scheme P most of the time, but scheme G (S) outperforms scheme GP (SP) most of the time. The simulation results support our conjectures about using PACE in frame-based real-time systems in [12]. Therefore, PACE is not recommended in the MEEC scheme.

Next, we compare the MEEC scheme with other schemes, all without using PACE. Figure 3 shows the maximum and average energy savings of our scheme over other schemes for both the synthetic CPU and XScale. From the figure we can see that the MEEC scheme achieves an average of 20.45% (up to 33.45%) energy savings over the next best scheme (proportional) for the synthetic CPU, and an average of 6.52% (up to 20.85%) energy savings over the next best scheme (statistical) for XScale. The energy savings are significant. The two key factors that affect the energy savings are the minimum speed of the CPU and the number of speeds available from the CPU. In computing the speed schedules, the MEEC scheme assumes unrestricted continuous frequency. Because of the convexity of the power function, high speed is not usually obtained by the MEEC scheme. But low speed is desired because the MEEC scheme can navigate the full spectrum of available speeds and can find the best speed that minimizes the expected energy consumption. The importance of the number of speeds available from the CPU is obvious given that we need to convert the continuous speeds to discrete speeds. Therefore, because the minimum speed of the synthetic CPU is less than that of XScale and the number of speeds of the synthetic CPU is greater than that of XScale, the energy saving for XScale is less than the synthetic CPU.

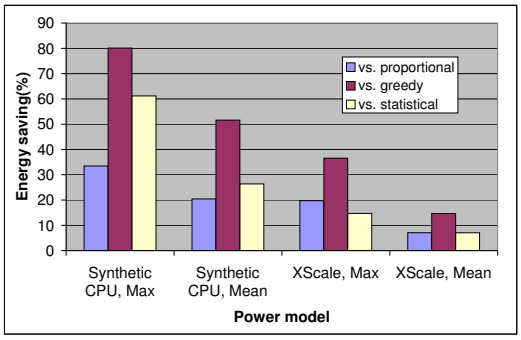


Figure 3. Energy savings of the MEEC scheme over other schemes for the synthetic workload

5.3 Automatic Target Recognition (ATR)

The ATR application[2] does pattern matching of targets in images. In ATR, the regions of interest (ROI) in one image are detected and each ROI is compared with all the templates. The number of target detections in each frame varies from 0 to 8 detections. Image processing time is proportional to the number of detections within an image.

In our system model, a front-end is responsible for collecting images and sending the images periodically to a back-end equipped with an Intel XScale CPU for target recognition. The back-end is required to finish processing all the images that it receives by the end of the period (frame) in order to process the next batch of images

[1] When the time allocated to execute a task is determined, use PACE technique to execute this task within the allocated time. The same holds for using PACE for other schemes.

[2] The original code and data for this application were provided by our industrial research partners

in a timely fashion. The period is 100 ms and the front-end sends 1 to 6 images to the back-end for one period.

Each task processes an image with 1 to 8 ROIs. We obtained the probability distribution of the workload of the task by profiling on a training image set, then precomputed the speed schedule (that is, computed the β_i values, see Section 4) for having $1, 2, 3, 4, 5, 6$ images to be processed in one period (frame), respectively. The six speed schedules are stored in the back-end. When a period begins, the back-end counts the number of images received and applies the corresponding speed schedule. Figure 4 shows the energy savings of the MEEC scheme over other schemes when the back-end has $1, 2, 3, 4, 5, 6$ images to process. From the figure we can see that the MEEC scheme can achieve an average of 11.04% energy savings (not counting the case for 1 image because all schemes achieve the same performance in this case) over the next best scheme (statistical).

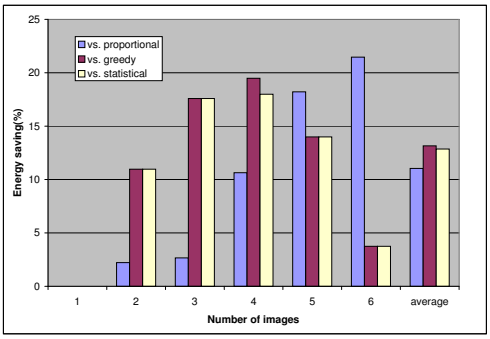


Figure 4. Energy savings of the MEEC scheme over other schemes for ATR

6 Conclusions

In this paper, we demonstrate and present extensive evaluations of the MEEC scheme proposed in [12]. We first review the existing DVS schemes and demonstrate the MEEC scheme through a simple example. Then we evaluate the existing DVS schemes and the MEEC scheme through different power models and different workloads. Evaluation results show that the MEEC scheme can achieve significant energy savings over the existing schemes. Another important conclusion from this work is the demonstration that using only static information or aggregating dynamic information, even with probabilistic techniques, will not produce as good results as when dynamic information for each task is considered separately.

Future work will investigate the case of the problem where different tasks have different deadlines.

References

[1] N. AbouGhazaleh, D. Mossé, B. Childers, R. Melhem, and Matthew Craven. Collaborative Operating System and Compiler Power Management for Real-Time Applications. In *RTAS*, May 2003.

[2] H. Aydin, R. Melhem, D. Mossé, and P. Mejia-Alvarez. Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems. In *RTSS*, pages 95–105, December 2001.

[3] R. Ernst and W. Ye. Embedded Program Timing Analysis based on Path Clustering and Architecture Classification. In *ICCAD*, San Jose, CA, 1997.

[4] I. Hong, G. Qu, M. Potkonjak, and M. Srivastava. Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors. In *RTSS*, Madrid, Spain, December 1998.

[5] W. Kim, D. Shin, H. Yun, J. Kim, and S. Min. Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems. In *RTSS*, December 2002.

[6] J. Lorch and A. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *ACM SIGMETRICS*, June 2001.

[7] J. Lorch and A. Smith. Operating system modifications for task-based speed and voltage scheduling. In *MobiSys*, May 2003.

[8] D. Mossé, H. Aydin, B. Childers, and R. Melhem. Compiler-Assisted Dynamic Power-aware Scheduling for Real-Time Applications. In *COLP*, October 2000.

[9] P. Pillai and K. G. Shin. Real-time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *SOSP*, pages 89–102, October 2001.

[10] C. Rusu, R. Xu, R. Melhem, and D. Mossé. Energy-Efficient Policies for Request-Driven Soft Real-Time Systems. In *ECRTS*, Catania, Italy, July 2004.

[11] Intel xscale microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[12] R. Xu, D. Mossé, and R. Melhem. Minimizing Expected Energy in Real-Time Embedded Systems. In *EMSOFT*, Jersey City, New Jersey, September 2005.

[13] R. Xu, C. Xi, R. Melhem, and D. Mossé. Practical PACE for Embedded Systems. In *EMSOFT*, Pisa, Italy, September 2004.

[14] F. Yao, A. Demers, and S.Shankar. A Scheduling Model for Reduced CPU Energy. In *FOCS*, pages 374–382, 1995.